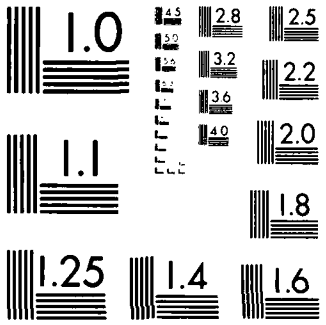
AD-A155 224 ADA COMPILER VALIDATION CAPABILITY: LONG RANGE PLAN(U) 1/1
SOFTECH INC WALTHAM MA J B GOODENOUGH ET AL. FEB 80
1067-1.1 MDA903-79-C-0687

UNCLASSIFIED F/G 9/2 NL

END
FILMED
DTIC


1.0
4.5
5.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

(1)

AD-A169 22[illegible]

[illegible]

SOFTECH

THE SOFTWARE TECHNOLOGY COMPANY

1

ADA COMPILER VALIDATION CAPABILITY: LONG RANGE PLAN

1067-1.1

February 1980

APPROVE[illegible]

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A/1	

Submitted to

Defense Advanced Research Projects Agency
Arlington, Virginia

Contract MDA 903-79-C-0687
Data Item A0002

Prepared by

SofTech, Inc.
460 Totten Pond Road
Waltham, MA 02154

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A155224	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Capability; Long Range Plan		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER 1067-1.1
7. AUTHOR(s) John B. Goodenough, John R. Kelly		8. CONTRACT OR GRANT NUMBER(s) MDA903-79-C-0687
9. PERFORMING ORGANIZATION NAME AND ADDRESS SofTech, Inc. 460 Totten Pond Road Waltham, MA 02154		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency IPTO - 1400 Wilson Blvd. Arlington, VA 22207		12. REPORT DATE February 1980
		13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Compiler validation, programming languages, Ada, compiler testing, software testing, compiler certification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Ada Complier Validation Capability (ACVC) will consist of tests, test documentation, and tools for determining to what extent Ada compilers conform to the language standard. This Report describes the functions to be supported by the ACVC, the general nature of the capabilities to be provided, and the planned approach for each of the development phases.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

The objective of the Ada Compiler Validation Capability (ACVC) is to determine to what extent an Ada compiler conforms to the Ada Standard. The purpose of this Report is to describe planned validation capabilities, their relation to other standard~~ar~~dization activities, and our planned development approach.

The ACVC is being developed under a three phase contract, with phases 2 and 3 being performed at the option of the Government. The periods of performance and principal objectives of each Phase are:

- Phase 1 (25 September 1979 through 1 October 1980) -- to produce a baseline set of validation tests, tools needed to support validation activities, and procedures for using the tests and tools;
- Phase 2 (2 October 1980 through 1 October 1981) -- to increase test set coverage while maintaining the baseline capability, and
- Phase 3 (2 October 1981 through 1 April 1982) -- to develop new validation approaches that attack difficult validation problems not fully solved in the earlier Phases. In addition, validation capabilities provided in earlier Phases will be updated and maintained. [illegible]

In the next Section of this report, we describe the functions to be supported by the ACVC and the general nature of the capabilities to be provided. In Section 3, we describe specific plans for each of the Phases. In Section 4, we specify risk areas. In Appendix A, we specify documentation standards to be applied to the ACVC.

SECTION 2

SYSTEM SUMMARY

In this Section, we describe the general nature of the capabilities needed for Ada compiler validation.

2.1. ACVC Objectives and Limitations

The primary purpose of the ACVC is to provide information the Government needs to decide whether Ada compilers (as well as other kinds of language processors) conform to the Ada language standard (as defined by the forthcoming Ada Report). A secondary, but nontheless important purpose is to assist Ada compiler developers in complying with the Ada standard and in improving the quality of their compilers. To achieve this purpose, the ACVC will contain written information of value to all compiler developers and will provide reports to aid developers in identifying and understanding errors discovered by the ACVC.

It should be noted, however, that the ACVC validation tests will not form a complete acceptance test for an Ada translator. Overall acceptability encompasses more than compliance with a language standard. Among the issues not addressed by the tests that are to comprise the ACVC are some that are of interest primarily to compiler procurers and some of interest primarily to compiler users. Among the issues of interest to those buying a compiler are:

- the compiler's maintainability, as reflected by its
  - internal design structure
  - the implementation standards followed by its source code
  - the quality of its design and implementation documentation
- its interface with other support tools, e.g., as reflected by
  - its object file format
  - its symbol table format and options for producing a binary symbol table for use by a symbolic debugger

SOFTECH

the internal format of the program library

Among the issues of interest to potential users of compilers are:

- the range of compiler options
- its compile-time efficiency
- the helpfulness of its error message format and content
- the efficiency of the object code it generates

Since information about these user-oriented issues is useful for improving the quality of Ada compilers, it should eventually be gathered by the ACVC.

2.1.1. Validation Test Effectiveness

The quality of a compiler validation capability is ultimately measured by the number and importance of the ways a validated compiler is later discovered to be nonconforming. If the baseline ACVC is of poor quality, Ada compilers containing significant errors may be approved for official use. Once such compilers are in use, it is often costly to make them standard-conforming. The net result is the development of implementation-defined Ada dialects, and the goal of a DoD Common Language will not have been achieved.

Consequently, our approach will apply several important design principles underlying the construction of high quality tests. Perhaps the most important is that tests must be designed so their successful execution implies (or at least makes it highly likely) that certain implementation errors are absent. The basis of this principle, which has been argued at length in [1] and [2], is that unless tests have been designed so their successful execution rules out certain errors, little is learned from such success. And of course, a validated compiler is one that has passed all its tests! This means that to construct an effective compiler validation capability, one must first hypothesize the kinds of errors implementers may make and then construct tests that can only succeed if the errors are absent.

But tests developed in advance of a compiler implementation, i.e., tests developed without knowledge of a compiler's internal structure (so-called "black box" tests) cannot, in principle, detect <u>all</u> errors a compiler may contain, even though, strictly speaking, any errors constitute non-conforming behavior.* Since not all errors can be detected, our objective in designing black box tests will be to ensure that the cost of developing and executing tests intended to detect certain errors is not greater than the benefit of detecting the error. For example, undetected errors are non-critical if they are unique to a particular application program or programmer and an alternative use of the language provides a satisfactory "write-around" until the error is fixed. Such errors usually reflect the use of rare combinations of language features. Moreover, although many implementation errors arise from failure to deal correctly with certain combinations of language features, we will not attempt to detect errors judged unlikely to occur in practice or whose effect is unlikely to be critical. To achieve this objective, the tests we develop will be based on an analysis of implementation problems and important interactions among language features that must be successfully dealt with by every conforming implementation. Tests will then be devised to check

---

* The possibility of developing tests based on knowledge of a compiler's internal structure will not be addressed at least until Phase 2 of our planned effort, and possibly not until Phase 3, since such tests are currently very difficult and costly to construct.

whether these difficulties have been dealt with properly.

2.1.2. Validation Support Tools

Since the validation tests are to be used for compilers executing in a variety of host environments and generating code for a variety of target environments, the tests must be easily adapted to cope with these differences. This means minimizing the amount of manual effort required to adapt tests to host and target environments and to analyze the results of test compilations. We will provide semi-automated tools to help reduce the manual effort involved in conducting these tests, and will design the tests to make use of such tools easier than might otherwise be the case.

2.2. ACVC Structure

The ACVC has three main components:

- An Implementers' Guide describing explicitly the implications of the Ada Report. It will specifically address implementation issues that might be overlooked by an unwary implementer. The purpose of this Guide is partly to point out the implications of statements made in the Ada Report and partly to specify conditions to be checked by validation tests.
- Test programs to be submitted to a compiler.
- Validation support tools that assist in preparing tests for execution and in analyzing the results of executions.

These components are described in more detail below.

2.2.1. The Implementers' Guide

The primary document to be produced in Phase 1 is the Implementers' Guide. This document describes the ramifications of the Ada Report and in particular, describes implementation issues that must be properly addressed in a production compiler design. The document also specifies objectives, constraints, and guidelines for developing specific validation tests. The document will be keyed to the subsections of the Ada Report. For each subsection, there will be an analysis treating the following topics:

1. Ramifications of the Semantics -- This section will document implications of the language semantics. In cases where the implied semantics are derived from statements in separate sections of the Ada Report, a rationale for the interpretation will be given to aid those who are not intimately familiar with the Report or the rationale underlying the Ada design. We will give particular attention to interactions between Ada constructs. For example, for LRM Section 5.1.1 (Array and Slice Assignments), we would point out that if A and B are strings with index ranges 1..10, then

   ```
   A(1..4) := A(2..5) & "";
   A(1..0) := "";
   A(1..0) := A(2..1);
   ```

   are all legal assignments. The first assignment does not raise the OVERLAP_ERROR exception since the result of concatenation is not a slice value, and hence there are no index ranges that overlap. The second assignment is legal because the number of elements (i.e., zero) is the same and there is no requirement that the bounds of an empty slice satisfy the range constraints for indexing elements of the array. The third example shows a case where OVERLAP_ERROR should not be raised even though the slices have a value in common.

2. Compile-time Constraints -- This section will explicitly state context-sensitive syntactic and semantics constraints to be checked (at compile-time or load-time), i.e., it will detail the implications of the legality constraints stated broadly in the Ada Report. Listing these constraints explicitly provides a convenient checklist for implementers to ensure they cover all the context-sensitive validity constraints implied by the specification and also, of course, suggests validation tests to be provided. When useful, we will note non-straightforward ways the constraints can be violated.

3. Exception Conditions -- This section will explicitly state what exception conditions may be raised, since these determine what run-time checks implementers must provide. For example, in the case of Section 3.3, we would note that RANGE_ERROR is the only exception condition explicitly associated with evaluation of constraints, and we will explain the circumstances under which this exception is raised for each form of constraint.

4. Test Objectives and Design Guidelines -- This section will state test objectives, give design guidelines for test case construction, list implementation errors to be checked for, and list problems to be kept in mind while designing or writing test cases (e.g., to ensure that the tests are as portable as possible and that they are effective in determining the actual behavior of a compiler). Borderline cases to be considered by an implementation will be noted here with program examples when such examples are the most straightforward way of illustrating the point being made. These examples are candidates for inclusion in the validation test set.

5. Gaps in the Analysis -- This section will document those aspects of the Ada specification for which test objectives have not yet been defined, either because of potential changes in the language design or because it is not clear how to construct tests that will have the desired properties. In essence, this list describes those aspects of the ACVC for which further work is needed, either later in Phase 1, or, if the problem is a research problem, it explains the issues for which further study is required. In addition, this section will list questions for which the specification does not provide obvious answers. Our initial analysis will concentrate on identifying such questions.

2.2.2. Validation Test Structure

One issue in developing a set of validation tests is whether the tests should be few in number, which implies a given test can fail for many reasons, or whether individual tests should focus on a small set of implementation errors, thereby leading to a larger number of tests to be processed by a compiler.

The advantage of having a large number of fairly specialized tests is that more information is obtained about compiler flaws. Although validations are in principle PASS/FAIL (i.e., either the compiler conforms to the language Standard or it doesn't), in practice, such a response to a validation is as unacceptable as a compiler that terminates compilation with the single message "PROGRAM IN ERROR." Just as compilers are designed to detect more than one error in a program, so it is useful for validation tests to reveal as many nonconforming aspects of a compiler as possible. There are several reasons for this. First, the compiler being validated may be needed for immediate use by some project; if the compiler's nonconformities are minor with respect to its anticipated use, DoD may decide it is cost-effective to permit use of the non-conforming compiler while the errors are being corrected. The question of whether such use should be permitted is a policy decision based on the nature of the nonconforming behavior and the expected use of the compiler. Such a policy decision should not be pre-empted by minimizing the information produced by the test set.

Second, if validation stops after detection of the first error or only a few errors, leaving much of the compiler untested, then these tests must be repeated for the corrected compiler version. If only a few errors are discovered each time, the cost of retesting becomes significant. Since the Government is likely to be paying for the validation cost (in one way or another), it is important to obtain as much information about nonconforming behavior as possible to reduce retesting costs.

Third, if the test modules are large, fatal compiler errors yield no information about what features have been successfully tested. To obtain more information from a fatal error, a large test program must be decomposed into self-contained modules. This can be costly and time-consuming if the tests were not originally designed to be decomposed into small units.

Thus, we propose to develop many small validation test modules (approximately 1000 programs containing an average of 50 lines of text). The

SECTION 4

RISK AREAS

The primary area of risk is that if the Preliminary Ada Reference Manual is found to be very difficult to interpret, a large number of questions will have to be asked of the language designers and answers received before analysis of implementation difficulties and test objectives can be undertaken productively. Our initial analysis of some of the seemingly simpler areas of the language indicates that this may pose a problem. Our plan is to identify such questions as quickly as possible, so work can proceed on those areas of the language where the intent is clear.

A secondary area of risk concerns the magnitude of the language revisions. The greater the change between the preliminary and final version of Ada, the less thorough will be the coverage we can provide in Phase 1. The worst situation would be for us to spend time analyzing areas of the language that are later so changed that most of the analysis is no longer applicable. Given sufficient visibility into the language revision process, this risk can be reduced, but the possibility of surprises is always present, at least until the language is frozen.

capability for validating Ada's tasking semantics and for ensuring that optimizations are performed correctly.

In addition, it is likely that the Ada standard will be updated during the Phase 3 period to reflect the experience gained in producing the first Ada production compilers and in using Ada for some embedded computer applications. The validation tests will be updated to reflect these changes as well as any deficiencies discovered in the course of validating various Ada compilers.

In Phase 1, we will also produce a basic set of tools for adapting tests to the requirements of a particular compiler, and we will provide a description of procedures to be followed in using these tools to validate a compiler.

3.2. Phase 2

The areas intentionally given reduced coverage in Phase 1 will be addressed more thoroughly in Phase 2. In addition, the baseline coverage will be updated and improved based on further analysis of the official Ada Report. Since the official Ada Report will at best appear toward the end of Phase 1, it is likely that further analysis in Phase 2 will show deficiencies in the Phase 1 test coverage that were not recognized at that time. These deficiencies will be corrected in Phase 2, and all associated documentation updated.

A key Phase 2 activity will be to use the baseline tests in validating an experimental or production compiler. Past experience with validation tests indicates that even when considerable care has been taken in test design, more problems will be discovered when the validation capability is used the first time than when it is used on subsequent occasions. Correcting these problems will be an important Phase 2 activity.

3.3. Phase 3

Phase 3 will address research issues to further refine the validation capabilities. Although it is difficult at this time to accurately predict what these issues will be, it is likely that work will continue to improve the

1980. An intermediate draft for the second iteration will be available in May 1980. A first draft for the third iteration (covering all of revised Ada) will be available in August 1980. The final version will be delivered at the end of Phase 1, on 1 October 1980. It is our intent to permit limited distribution of these intermediate drafts to people designated by DARPA and to hold reviews at which recipients of the drafts will be invited to comment. In this way, we will identify errors and omissions in our analysis and the validation effort will benefit from the experiences of those engaged in experimental or production implementations of Ada compilers.

Not all areas of Ada can be covered with equal thoroughness in the baseline tests. In particular, more tests are needed to the extent an implementer has many ways of implementing a required capability. The greater the variety of implementation approaches and requirements, the greater the number of tests needed to check that each approach has been carried out successfully. There are several areas where Ada leaves considerable implementation flexibility. These areas will not be covered thoroughly in Phase 1. They are:

- tasking semantics for a wide range of target computer architectures;
- the INPUT_OUTPUT package for the full range of I/O peripherals Ada compilers might support;
- the correctness of optimizations
- other standard packages that may eventually be defined, e.g., the mathematical functions package.

Only basic capabilities in these areas will be covered in Phase 1.

SECTION 3

PLANS FOR EACH PHASE

In this Section, we describe our current plans for capabilities to be provided in each Phase of the effort.

3.1. Phase 1

The purpose of Phase 1 is to produce a baseline validation capability that addresses all aspects of Ada to some extent and whose weaknesses in coverage are documented. A prime focus in Phase 1 will be the Implementers' Guide. A high quality validation test set (i.e, one that detects even subtle nonconformities) must address interactions among language features (since these are a primary source of implementation problems and subtleties) and must cover potential specification misinterpretations and subtle ramifications. Pointing out implementation problems explicitly will be of considerable assistance in discouraging Ada dialects. Often dialects arise because of inadvertent misunderstanding of a specification. If the Implementers' Guide points out possible mistakes, implementers are less likely to make them. Hence, producing a thorough Implementers' Guide is necessary not only to guide the construction of high quality tests but also to help prevent inadvertent deviations from the specification.

The approach we are taking in Phase 1 is to make three iterations in producing the Implementers' Guide. The iterations are needed because Ada is currently undergoing revision. To cope with the possibility of changes, the first iteration will address only the most stable parts of Ada. A draft Implementers' Guide for the stable subset of Ada will be available in March

2.3.1.5. Archiving Findings

As a last step, all plans, tests, preparation aids, post processing analyzers, and findings are archived for later use when a revised version of the compiler may need to be revalidated. The ACVC will recommend archiving procedures and concepts.

2.3.2. Revising the Validation Capabilities

In essence, the validation capabilities consist of:

- the validation tests
- the Implementers' Guide
- validation support tools
- descriptions of procedures for using the tools

There are a variety of reasons for modifying any of these capabilities:

- problems encountered in conducting a validation may lead to modifications in the validation support tools and/or the test cases (to remove test case errors).
- errors discovered in validated compilers may indicate previously undocumented implementation problems; both the Implementers' Guide and the validation tests must be updated.
- ambiguities are discovered in the language specification. Additional Class E tests are needed to determine how a given implementation has resolved these ambiguities.
- the validation tests are expanded to cover known weaknesses.
- the language specification is modified.

After the Baseline Validation Capability is delivered, it will be placed under configuration control. Procedures will be specified for producing new releases of the tests, tools, and associated documentation (see Appendix A).

2.3.1.3. Executing Validation Tests

The files prepared in the previous step are transported to the compiler host and compiled. All object code except for Class C, E, and F (and possibly, A) tests is discarded. Listings from all compilations are saved in machine readable form and processed on the validator host computer to check that all intended tests were compiled and that expected results were achieved.

The object code is executed on the target computer and results are saved in machine readable form for automatic analysis on the validator host. Since people are notoriously unreliable in scanning results for correctness, extensive effort will be directed during test case design to ensure results can be automatically checked insofar as possible.

Any difficulties encountered during compilation or execution due to errors in the tests themselves will be noted and used as inputs to the Test Revision process.

2.3.1.4. Summarizing Results

Using editing tools and report templates provided by the ACVC, results of a validation will be summarized, following the report structure referenced in Appendix A. The purpose of the Validation Summary Report is to concisely describe what compiler was validated and what nonconforming behaviors were discovered. Since groups of tests may fail for a common cause, preparation of a validation summary report is not a completely automatable task. Results must be studied to extract as much useful information as possible.

In general, the following steps must be performed before the Ada Validator test files are transported to the compiler host:

- generate specific versions of tests for the full range of capabilities supported by the compiler (e.g., to test all levels of numeric precision supported, including interactions among these levels);

- substitute implementation-dependent parameters in appropriate tests, e.g., literal values testing the upper and lower limits of the numeric ranges supported, string literals that contain all characters of the supported character set, etc. In general, the validation tests will contain the equivalent of text macro invocations so implementation-dependent constructs can be substituted at the appropriate places.

- embed the tests in the appropriate job control language for compiling, linking, and executing the tests.

- generate file names that satisfy the compiler host's naming conventions.

- tailor Class D tests to the correct size.

- prepare text processing macros suitable for analyzing machine-processable results of compilation and execution.

A file of implementation-dependent parameters must be prepared for every validation. This file gives values for numeric precisions, values of attributes of built-in types, and other parameters to be used in generating tests that are appropriate for a specific compiler. An important aspect of the ACVC test set design is identifying places where these parameters should be used in specific tests. Unless these parameters are used appropriately, it will be difficult to adapt the tests to the capabilities of a specific compiler.

Difficulties in using the test preparation editing tools will be noted and input to the Tool Revision process.

compiler, the method of transferring object code for execution on the target computer, the methods for linking and loading modules to be executed, the method for collecting and displaying compilation results, etc.

- the object host environment, including its file structuring capabilities, file naming conventions, the method for linking and loading (if this is to be done on the target computer), the method of initiating object code execution, the method for collecting execution results in machine-processable form, etc.

In addition, if the validation is being performed on a new version of a previously validated compiler, information from the previous validation must be analyzed to determine how revalidation costs might be minimized (e.g., by combining test programs into larger modules, by reusing test preparation macros, etc.).

This information is used to prepare a comprehensive Test Plan describing what tests are to be run, information needed to prepare tests for submission to the compiler, estimates of time required to perform the tests, procedures for manually and automatically analyzing test results, etc. (see Appendix A). The Test Plan controls subsequent phases of testing activities. The ACVC will include a prototype Test Plan to serve as a guide for creating actual Test Plans.

2.3.1.2. Preparing Tests for Submission

The basic purpose of this activity is to adapt, and in some cases, generate, text files in a form suitable for submission to the compiler being validated. A number of ACVC text editing macros (and macro templates) will be provided to automate as much of the preparation process as possible.

tests, tools, and procedures. In this Section, we describe the general nature of these activities since they affect the structure and content of the ACVC.

2.3.1. Validating a Compiler

There are five principal activities performed in validating a compiler. These are:

1. Collecting information needed to select and adapt the validation tests for the compiler being tested.

2. Preparing the tests for submission to the compiler.

3. Performing the tests, i.e., compiling them, executing the executable tests, and collecting the results for later analysis.

4. Summarizing the test results.

5. Archiving the findings.

2.3.1.1. Collecting Information

The principal output of this activity is a Test Plan describing all the information needed to prepare and conduct the validation. Information is needed regarding:

- the language accepted by the compiler, e.g., the character set accepted by the compiler (full ASCII, EBCDIC, 63-character CDC, etc.), allowed length of input lines, how an "end-of-line" is defined, the semantics and restrictions on representation specifications (Section 13 of the Preliminary Ada Reference Manual), the syntax and semantics of commands for manipulating the program library (Section 10.4), any language subsetting, the number of levels of integer, fixed, and floating point precision supported and their names, ranges associated with numeric types, etc. This information is used in selecting and adapting the tests to be used, e.g., some tests do not apply if only a single integer precision is supported.

- compiler options provided, which ones are to be tested, and how they are exercised.

- the compiler host environment, including file structuring capabilities, file naming conventions, the method of invoking the implementation dependent (e.g., the maximum and minimum integer literal values). Appropriate values must be inserted into some tests.

All these text handling problems will be dealt with by the validator host tools, which will consist primarily of text editing tools and macros. We plan to use TECO and TECO macros to perform commonly needed text manipulations. If the compiler host is available on the ARPANET, the validation tests, once adapted to the CH requirements, will be sent to the CH using the Net. If the CH is not on the Net, the tests will be brought to the host using an appropriate physical medium. If a comparable ARPANET host is available, it might be used to transfer the text to such a medium.

Text editing tools will also be used to analyze the machine processable results of compilations and executions -- to confirm that all required tests were submitted and that all test failures are identified.

The text processing tools we provide will be documented in accordance with the specifications provided in Appendix A.

2.2.4. Validation Procedures

Usage of the tests and tools will be described in a Validation Procedures Manual structured in accordance with the specification cited Appendix A. The general nature of these procedures is discussed in Section 2.3.

2.3. Validation Activities to be Supported

There are two principal activities supported by the Ada Compiler Validation Capability: 1) compiler validation and 2) revising the validation

Additional test Classes may be added as a result of Phase 2 and Phase 3 efforts, e.g., tests of object code efficiency, compiler performance, etc.

Within each class, a test will be given a name corresponding to the Chapter, Section, and Subsection number of the part of the Report to which the test pertains. Since most subsections require more than one test, tests will be assigned sequential numbers within subsections. Each source code module will be in a separate file which will be given the name of the test. There may also be a number of INCLUDE files used in conjunction with test modules.

2.2.3. Validation Support Tools

There are conceptually three environments relevant to validating a compiler:

- the validation host (VH) environment, in which validation tests are prepared for execution by an Ada compiler and validation results are analyzed;
- the compiler host (CH) environment, in which Ada tests are compiled. In some implementations, target modules may also be loaded in the CH environment; executable code is then transported to the object computer for execution;
- the object host (OH) environment (the target computer) in which Ada programs are executed.

An important aspect of our approach to the ACVC is the Validator Host concept. The Ada validation tests will constitute a fairly large amount of textual material. Powerful text editing and file management tools are needed to create, access, and maintain this material. More importantly, when preparing to perform a compiler validation, the tests must be adapted to conform to the conventions of the Compiler Host for submitting text to the compiler and obtaining output. In addition, a variety of test parameters may be

- Class L Tests. These tests also demonstrate the rejection of illegal Ada programs, but the errors in these tests may be detected at link-edit or load time by some implementations. These tests are grouped together because their PASS/FAIL criterion is different from that for Class B tests, namely the test succeeds if an error is indicated before execution begins. An example of a class L test would be compiling a version of a package with revised subprogram interfaces and then attempting to load the package without first recompiling and/or modifying the units using the revised package.

- Class C Tests. These tests are designed to be executed after compilation. They verify that legal Ada programs are executed according to the specified semantics. The programs are self-checking and generate PASS/FAIL messages before execution is terminated.

- Class D Tests. These tests are used to check capacity requirements for Ada compilers, e.g., that the number of subscripts, parameters, parenthesis nesting levels, number of link-edit symbols, etc. is only limited by host system capacities rather than arbitrarily. When a compiler does impose small capacity limits, the tests are designed to indicate approximately what capacities the compiler supports (e.g., at least seven but not fifteen subscripts).

- Class E Tests. These tests examine the permissive aspects of the Ada Report, e.g., whether a warning is issued if the compiler detects expressions guaranteed to raise an exception if the code is executed (see Section 10.6 of the Preliminary Ada Reference Manual), e.g., as in X := A/0.0;. Such expressions can arise as a result of generic instantiation or inline substitution of subprograms. Other aspects of an implementation that affect its interface with the external object computer environment but which are not currently specified are also checked by these tests, e.g., consider

```
type BIASED is new INTEGER range 0..15;
for BIASED use 4;
```

  In interfacing with external devices, it could be important to know whether 2#1000 represents the value zero (if a signed biased representation is used) or the value 8 (if an unsigned representation is used). Other tests in this class will be used to determine what interpretation an implementation has taken with respect to unintended ambiguities discovered in the Ada Report. The results of these tests are used to inform compiler users and to improve the Report.

- Class F Tests. These are tests that demonstrate the correct and complete operation of standard packages, e.g., the INPUT_OUTPUT package, the mathematical library package (when one is specified as a standard), etc.

disadvantage of having so many small tests is that submitting many small compilations is more costly than submitting fewer and larger compilations, and the job of keeping track of a large number of tests and their results is more demanding. However, an advantage of Ada is that separately compiled units can usually be combined together into a single compilation unit by enclosing them with a module body. Thus the benefits of submitting large compilation units can be achieved if tests are originally designed as small independent units, but the reverse is not necessarily the case. Thus, we plan to produce a large number of independently compilable validation tests, and provide plans and procedures for grouping them into larger units when this is appropriate (e.g., when retesting an upgraded or corrected compiler). In addition, we will provide validation support tools (see Section 2.2.3) that will ease the burden of dealing with a large number of tests.

The validation tests will be divided into seven classes:

- Class A Tests. These tests demonstrate that legal Ada programs are accepted by the compiler being tested. The criterion for passing these tests does not depend on execution of code and cannot be checked by executing object code (see Class C tests below). Successful compilation implies the generated code will execute successfully and print a PASS message. Hence, execution is not required, but can be performed if desired.

  Tests in this class may cause warning messages to be issued by the compiler, but must not generate error messages.

  There are rather few tests in this class. Among the Ada capabilities tested by Class A tests are the INCLUDE, LIST, and PAGE pragmas. In addition, tests designed to check whether an implementation has extended the list of reserved words would fall in this class.

- Class B Tests. These tests demonstrate that the compiler rejects illegal Ada programs. Class B tests are successful if they generate error messages at compile time. Some tests may generate multiple error messages, although strictly speaking, once a compiler has determined that a submitted program does not conform to the Ada Report, the Report is silent regarding what further errors the compiler must detect.

SECTION 5

REFERENCES

[1] Goodenough, J. B. and Gerhart, S. L., "Toward a Theory of Test Data Selection," IEEE-TSE-SE 1, 2 (June 1975), 156-173.

[2] Goodenough, J. B., "A Survey of Program Testing Issues," in Research Directions in Software Technology, P. Wegner, ed., MIT Press, Cambridge, Massachusetts 1979, 316-342.

APPENDIX A

Documentation Formats

This Appendix describes the formats to be used in documenting the following validation capabilities in each Phase of the effort:

- the Implementers' Guide
- validation support tools
- test programs
- validation procedures

The content of the Implementers' Guide will follow the structure indicated in Section 2.2.1 of this document. It will be formally updated in Phases 2 and 3 of this effort.

The validation support tools (i.e., the text editing macros) will be documented in accordance with DoDI 7935.1-S, dated 13 September 1977, using the specification for a Program Maintenance Manual.

The set of test programs will also be documented in accordance with DoDI 7935.1-S, using the specification for a Program Maintenance Manual. However, sections 2.4c-h and 3 do not apply. The functional description of each test program (Section 2.4b) will reference the appropriate test objectives section of the Implementers' Guide.

Procedures for using the validation support tools and tests to validate a compiler will be described using DI-M-3410, Users Manual (Computer Program). As part of these procedures, a sample template for a Test Plan will be provided, following the general outline of DI-T-3703A, CPCI Test

Plans/Procedures.

The format of the Validation Summary Report will be adapted from the format currently used for COBOL Validation Summary Reports provided by the Federal COBOL Compiler Testing Service.